Lab 2 JAE

Roi Naveiro y David Ríos Insua

22/06/2021

Cargamos los paquetes necesarios

```
library(randomForest)
```

```
## randomForest 4.6-14
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
library(keras)
```

Funciones auxiliares

- show_digit: Hace una gráfica del dígito en cuestión.

```
show_digit = function(img){
  img = t( apply(img, 2, rev) )
  image( img )
}
```

Lectura de Datos

Carga los datos de train y test en memoria

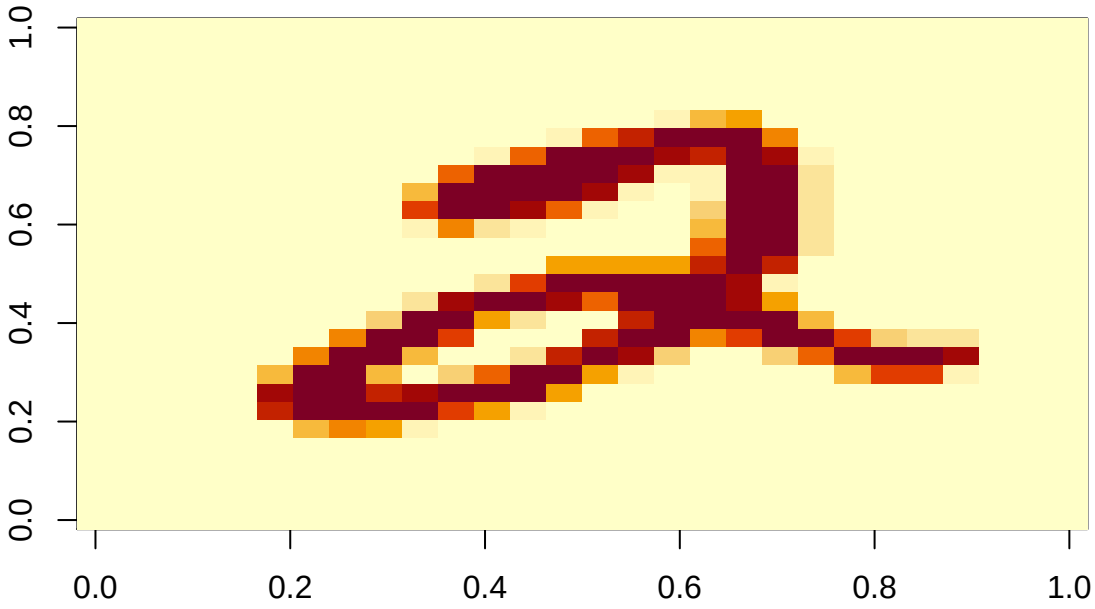
```
mnist <- dataset_mnist()
x_train <- mnist$train$x
y_train <- mnist$train$y
x_test <- mnist$test$x
y_test <- mnist$test$y
```

Visualiza algún ejemplo

```
y_train[6]
```

```
## [1] 2
```

```
show_digit(x_train[6,,])
```



Preprocesado de los datos

Antes del entrenamiento, es necesario aplanar los datos.

```
# remodelado
dim(x_train) <- c(nrow(x_train), 784)
dim(x_test) <- c(nrow(x_test), 784)
```

También es necesario pasar las etiquetas a la notación OHE.

```
y_train <- to_categorical(y_train, 10)
y_test <- to_categorical(y_test, 10)
```

Definición y entrenamiento de un modelo de regresión logística regularizada

Define un modelo de regresión logística con 10 outputs.

```
model_lr <- keras_model_sequential()
model_lr %>%
  layer_dense(units = 10, input_shape = c(784), activation = "softmax")
```

¿Cuántos parámetros entrenables tiene?, ¿por qué?

```
summary(model_lr)
```

```
## _________________________________________________________________________
## Layer (type)                 Output Shape              Param #
## =========================================================================
## dense (Dense)                (None, 10)                7850
## =========================================================================
## Total params: 7,850
## Trainable params: 7,850
## Non-trainable params: 0
## _________________________________________________________________________
```

Definir la entropía cruzada como función de coste y rmsprop como optimizador.

```
model_lr %>% compile(
  loss = "categorical_crossentropy",
  optimizer = optimizer_rmsprop(),
  metrics = c("accuracy")
)
```

Entrena el modelo con 30 épocas. Fija el tamaño de batch a 128. Además, escoge utiliza el 20% del conjunto de entrenamiento para la validación.

```
history <- model_lr %>% fit(
  x_train, y_train,
  epochs = 30, batch_size = 128,
  validation_split = 0.2
)
```

Parece que se estanca y no aprende gran cosa. ¿Se te ocurreo alguna solución?

```
# rescale
x_train <- x_train / 255
x_test <- x_test / 255
```

Una vez reescalamos...

```
history <- model_lr %>% fit(
  x_train, y_train,
  epochs = 30, batch_size = 128,
  validation_split = 0.2
)
```

Veamos cómo mejorarlo con una red profunda.

Definición y entrenamiento de la red Neuronal profunda con regularización L2

Definir una arquitectura de red que mapee el input a una capa densa con 256 unidades ocultas con activación tipo relu. La salida de estas capas ha de ser mapeada a otra capa densa con 128 unidades, también con activación relu. Finalmente, esta capa mandará señal a la capa final con 10 unidades y activación softmax para así recuperar probabilidades. Incluye regularización L2 en las dos primeras capas, con parámetro 0.001.

```
model_l2 <- keras_model_sequential()
model_l2 %>%
  layer_dense(units = 256, activation = "relu", input_shape = c(784), kernel_regularizer = regularizer_
  layer_dense(units = 128, activation = "relu", kernel_regularizer = regularizer_l2(l = 0.001) ) %>%
  layer_dense(units = 10, activation = "softmax")
```

Resumen del modelo

```
summary(model_l2)
```

```
## _________________________________________________________________________
## Layer (type)                 Output Shape              Param #
## =========================================================================
## dense_3 (Dense)              (None, 256)               200960
## _________________________________________________________________________
## dense_2 (Dense)              (None, 128)               32896
## _________________________________________________________________________
## dense_1 (Dense)              (None, 10)                1290
```

```
## ================================================================================
## Total params: 235,146
## Trainable params: 235,146
## Non-trainable params: 0
## ________________________________________________________________________________
```

¿Cuál es el número total de parámetros entrenables en este caso?

Definir la entropía cruzada como función de coste y rmsprop como optimizador.

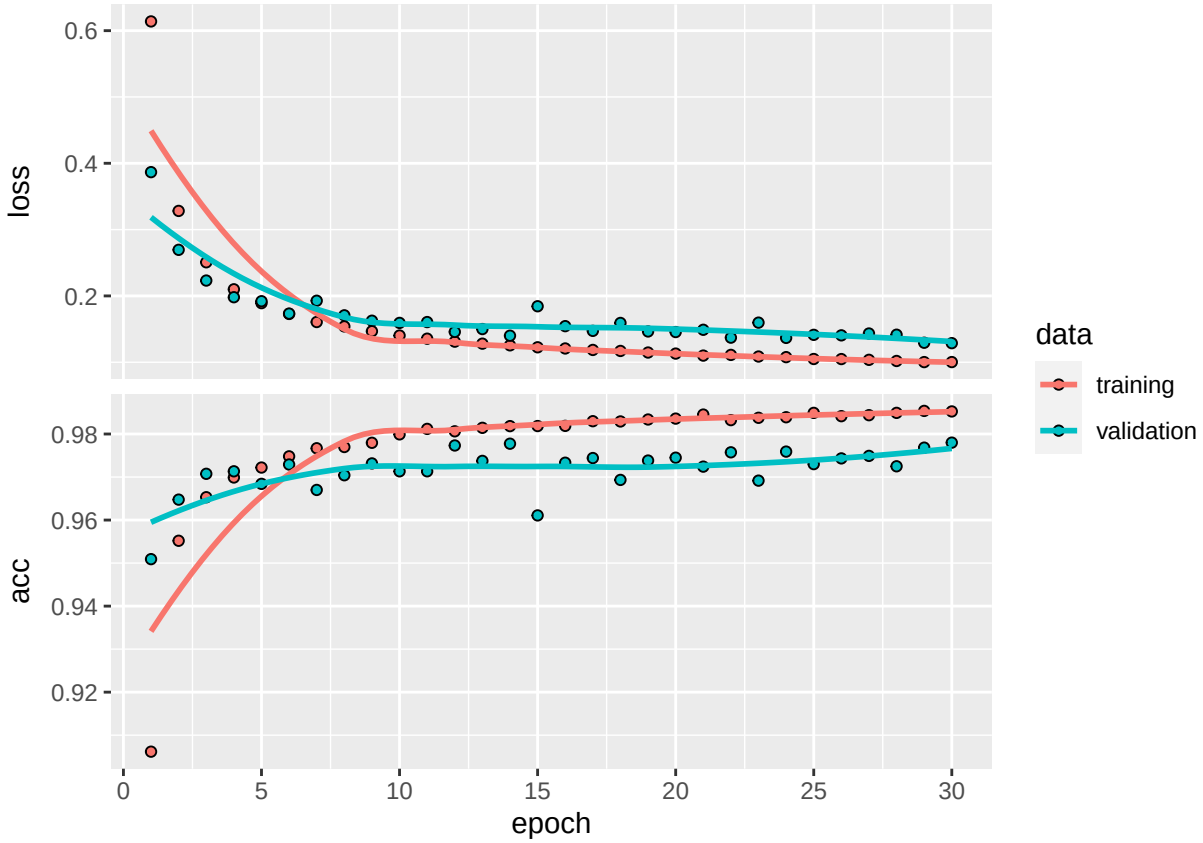
```
model_l2 %>% compile(
  loss = "categorical_crossentropy",
  optimizer = optimizer_rmsprop(),
  metrics = c("accuracy")
)
```

Entrena el modelo con 30 épocas. Fija el tamaño de batch a 128. Además, escoge utiliza el 20% del conjunto de entrenamiento para la validación.

```
history <- model_l2 %>% fit(
  x_train, y_train,
  epochs = 30, batch_size = 128,
  validation_split = 0.2
)
```

```
plot(history)
```

```
## `geom_smooth()` using formula 'y ~ x'
```



```
model_l2 %>% evaluate(x_test, y_test,verbose = 0)
```

```
## $loss
## [1] 0.1232754
##
## $acc
## [1] 0.9786
```

Definición y entrenamiento de la red Neuronal profunda con regularización dropout

Entrena la misma red que en el caso anterior. Esta vez, en lugar de utilizar regularización L2, utiliza dropout con proporción 0.4 en la primera capa y 0.3 en la segunda.

```
model <- keras_model_sequential()
model %>%
  layer_dense(units = 256, activation = "relu", input_shape = c(784)) %>%
  layer_dropout(rate = 0.4) %>%
  layer_dense(units = 128, activation = "relu") %>%
  layer_dropout(rate = 0.3) %>%
  layer_dense(units = 10, activation = "softmax")
```

Definir la entropía cruzada como función de coste y rmsprop como optimizador.

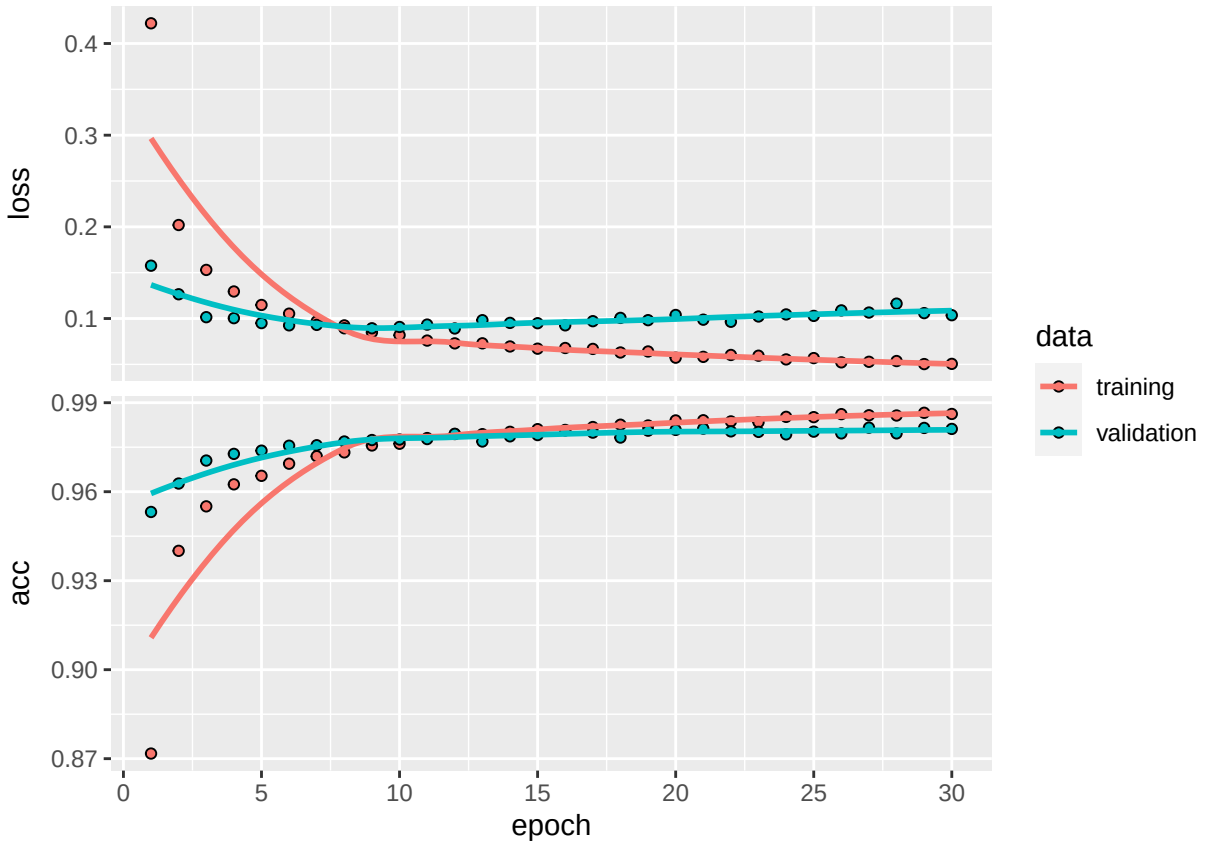
```
model %>% compile(
  loss = "categorical_crossentropy",
  optimizer = optimizer_rmsprop(),
  metrics = c("accuracy")
)
```

Entrena el modelo con 30 épocas. Fija el tamaño de batch a 128. Además, escoge utiliza el 20% del conjunto de entrenamiento para la validación.

```
history <- model %>% fit(
  x_train, y_train,
  epochs = 30, batch_size = 128,
  validation_split = 0.2
)
```

```
plot(history)
```

```
## `geom_smooth()` using formula 'y ~ x'
```



Evaluación y predicción en conjunto de test

¿Qué accuracy encuentras en el conjunto de test?

```
model %>% evaluate(x_test, y_test,verbose = 0)
```

```
## $loss
## [1] 0.1016975
##
## $acc
## [1] 0.9818
```

Guardar y leer modelos

Guarda el modelo creado en un fichero llamado "mnist_weights.hdf5". Lee el modelo de nuevo y realiza predicciones sobre el conjunto de test.

```
save_model_hdf5(model, filepath = "mnist_weights.hdf5", overwrite = TRUE,
  include_optimizer = TRUE)

new_model = load_model_hdf5("mnist_weights.hdf5", custom_objects = NULL, compile = TRUE)
```

```
preds = new_model %>% predict_classes(x_test)
```

Otros modelos

Aquí podemos probar otros modelos de los vistos en clase. Como ejemplo usamos Random Forest. Primero pasaremos las matrices de train y test a dataframe

```
train = data.frame(x_train)
train$y = as.factor(mnist$train$y)
#
test = data.frame(x_test)
test$y = as.factor(mnist$test$y)
```

Ahora entrenamos el modelo. Usaremos solo 1000 ejemplos de entrenamiento. La implementación de Random Forest de R no permite minibatches, con lo que usar todo el conjunto de entrenamiento sería muy costoso computacionalmente.

```
fit_rf = randomForest::randomForest(y ~ ., data = train[1:1000, ])
fit_rf$confusion
```

```
##    0   1  2  3  4  5  6   7  8  9 class.error
## 0 93   0  0  0  0  1  1   1  1  0  0.04123711
## 1  0 112  1  0  1  1  0   1  0  0  0.03448276
## 2  2   3 82  1  2  0  1   6  1  1  0.17171717
## 3  2   1  1 78  1  4  1   3  0  2  0.16129032
## 4  0   0  0  0 98  0  4   1  0  2  0.06666667
## 5  1   0  1  3  1 81  2   1  0  2  0.11956522
## 6  2   1  0  0  4  0 86   0  1  0  0.08510638
## 7  1   4  0  0  3  0  0 108  0  1  0.07692308
## 8  0   3  0  5  0  3  1   0 74  1  0.14942529
## 9  2   0  1  2  6  0  1   3  1 84  0.16000000
```

```
test_pred = predict(fit_rf, test)
mean(test_pred == test$y)
```

```
## [1] 0.8923
```

```
table(predicted = test_pred, actual = test$y)
```

```
##           actual
## predicted    0    1    2    3    4    5    6    7    8    9
##         0  960    0   11    6    1   14   17    1    8    7
##         1    0 1114    4    2    2   14    5   13    2    7
##         2    0    4  921   27    2    5   23   22   14    9
##         3    0    1    9  830    0   12    0    1   15    7
##         4    2    0   18    2  844   19   34    8   14   38
##         5    5    2    2   94    2  756   21    1   34   17
##         6    7    2   16    0   25   14  852    1   14    4
##         7    1    1   33   24    2   11    1  943   10   24
##         8    4   11   16   16    6   14    4    4  809    2
##         9    1    0    2    9   98   33    1   34   54  894
```

¿Cómo se comporta random forest respecto a las redes neuronales?